Graafinen käyttöliittymä lintujen rengastusjärjestelmään

Suunnitteludokumentti

Versiohistoria:

0.1	28.9.2000	Ensimmäinen luonnos.
0.2	17.10.2000	Toinen luonnos, moduulikuvauksia aloitettu.
1.0	19.10.2000	Ensimmäinen kokonainen julkaisu.
1.1	27.10.2000	Korjattu tarkastustilaisuudessa 24.10.2000 käsiteltyjä virheitä ja puutteita. Testaussuunnitelma lisätty.
1.2	9.11.2000	Jäädytetty versio. Viimeiset muutokset tehty.

Helsinki, 9. marraskuuta 2000

Lincoyan Kekki, Jaakko Kiviluoto, Juha Markkanen,
Juha Niemi, Joona Palaste, Janne Pasanen

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Ohjelmistotuotantoprojekti Tipu4

Sisältö

1 Johdanto **1**

1.1 Tuotteen tausta ja tarkoitus 1

1.2 Erikoissanasto ja käytetyt lyhenteet 1

1.3 Yleiskatsaus dokumenttiin 2

2 Sovellusalue **3**

2.1 Koko järjestelmä ja ohjelmiston rooli siinä 3

3 Vaatimusmäärittely ja siihen tehdyt muutokset **4**

4 Moduulikuvaukset **5**

4.1 Ohjelmiston toimintaperiaate 5

4.1.1 Ratkaisun filosofia 5

4.1.2 Toimintaperiaate 6

4.2 Moduulit ja niiden väliset suhteet 6

4.3 Tietokantayhteys 7

4.3.1 Yhtäaikaisuus 8

4.3.2 Luokka `tipu.db.ConnectionPool` 9

4.3.3 Luokka `tipu.db.Table` 13

4.3.4 Luokka `tipu.db.DatabaseOperation` 15

4.3.5 Luokka `OperationResults` 17

4.3.6 Luokka `tipu.db.SearchOperation` 18

4.3.7 Luokka `tipu.db.SearchResults` 20

4.3.8 Luokka `tipu.db.InsertOperation` 22

4.3.9 Luokka `tipu.db.UpdateOperation` 23

4.3.10 Luokka `tipu.db.DeleteOperation` 25

4.4 Servlet-luokat 26

4.4.1 Luokat `tipu.servlet.*` 27

4.5 Templatetiedostot 28

4.6 Apuluokat 28

4.6.1 Luokka `tipu.general.TipuProperties` 28

4.6.2 Luokka `tipu.general.Exec` 30

5 Käyttöliittymä 31

5.1 Yleistä 31

5.2 Navigointi 31

5.2.1 Valikko 31

5.2.2 Lomake 32

5.2.3 Painikkeet 33

5.2.4 Linkit 34

5.2.5 Virheilmoitukset 34

5.3 Esimerkki käyttötapauksesta 34

5.4 Tiedon haku 35

5.4.1 Kokonaisten tietueiden haku 35

5.4.2 Yksittäisten kenttien arvojen haku 38

6 Rajoitteet suunnittelulle ja toteutukselle 40

6.1 Noudatettavat standardit 40

6.2 Muut tarvittavat ohjelmat 40

6.3 Arvio toteutuksen koosta 40

6.4 Ohjelmointityyli 40

7 Testaussuunnitelma **41**

7.1 Testimoottori 41

7.2 Komponenttitestaus 41

7.3 Integrointitestaus 41

7.4 Validointitestaus 42

7.5 Hyväksymistestaus 42

Viitteet **43**

Kuvat

1 Yleiskuva ohjelmiston moduuleista ja niiden välisestä kommunikaatiosta. 7

2 Tietokantaoperaatioiden luokkarakenne ja tulostyypit. 8

3 Yksinkertaistettu esimerkki tietokantaluokkien käytöstä servletin suorittamassa poisto-operaatiossa. 9

4 Ruudun yläreunassa sijaitseva valikkopalkki. 32

5 Alavalikko rengastustoiminnoista. 32

6 Tyypillinen näkymä käyttäjän selatessa kantaan talletettujen kuntien tietoja. 33

7 Hakutapahtuman kulku. Pääsivusta on esimerkkinä kuva 8, hakukriteereistä kuva 9 ja tulossivusta kuva 10. 36

8 Kuntien ylläpitolomake. Käyttäjä pääsee Hae-painikkeella hakemaan kuntia. 36

9 Kuntien hakulomake. Lomakkeella on suoritettu haku kaikista Uudenmaan kunnista ja tulokset ovat listattuna hakukenttien alla. 37

10 Hakutulosten selaus tietueittain. 38

11 Rengastustietolomakkeessa joillakin kentillä on oma hakupainikkeensa. 39

12 Mikäli esim. Kunnan lyhennettä ei muista, voi arvoa hakea kenttäkohtaisella hakutoiminnolla. 39

1 Johdanto

1.1 Tuotteen tausta ja tarkoitus

Helsingin yliopiston eläinmuseon rengastustoimistossa käytetään SQL*Forms sovelluskehittimellä toteutettua käyttöliittymää rengastus ja tapaamistietoja sisältävään tietokantaan. Oracle on kuitenkin ilmoittanut, ettei se enää tue SQL*Forms-sovelluksia. Rengastustoimisto on tehnyt päätöksen korvata vanha käyttöliittymä uudella.

Projektin tarkoituksena on suunnitella ja toteuttaa graafinen käyttöliittymä lintujen rengastusjärjestelmään. Tavoitteena on korvata vanha merkkipohjainen käyttöliittymä graafisella. Tuotantokäytössä olevasta järjestelmästä on tarkoitus käyttää hyväksi Fortran-kielisiä ohjelman osia.

1.2 Erikoissanasto ja käytetyt lyhenteet

Connection pool Ohjelmistoratkaisu, jolla voidaan rajata yhteydet johonkin tiettyyn resurssiin. Näitä yhteyksiä luodaan ja jaetaan asiakkaille dynaamisesti.

HTML Hypertext Markup Language on internetissä käytetty tiedon esitystapa, jota asiakaspään selaimet osaavat tulkata.

HTTP Hypertext Transfer Protocol on WWW-liikenteen protokolla.

JDBC-ajuri Java DataBase Connectivity -ajuri, mahdollistaa tietokantakutsut palvelinsovelmista.

Kontrolli Rengastajien suorittama rengastetun linnun tapaaminen. Kontrollit tehdään pesiltä lintuja pyydystämällä tai esimerkiksi kiikaroinnilla.

Löytö	Ennestään rengastetun linnun kohtaaminen. Lintu voi olla elävä tai kuollut.
Rengastaja	Lintuharrastaja joka on oikeutettu ja koulutettu merkitsemään lintuja virallisilla linturenkailla.
Rengastus	Uuden renkaan laittaminen linnulle, jolla ei ennestään ole rengasta.
Servlet	Javalla koodattu ohjelmamoduuli, joka suoritetaan palvelimella, palvelinsovelma.
SQL	Structured Query Language. Standardi kieli, jolla voidaan määrittää erilaisia tietokantaoperaatioita.
Tapaaminen	Ennestään rengastetun, elävän linnun kohtaaminen.
Template	Dokumentin runko, joka sisältää tietyllä tavalla merkittyjä kohtia, joihin voidaan ohjelmallisesti lisätä vaihtuvia arvoja.
Tipukanta	Rengastustoimistolla käytettävä relaatiotietokanta rengastusten ja niihin liittyvien tietojen säilyttämiseen.
WWW-Tipu	Tässä dokumentissa kuvattu graafinen käyttöliittymä lintujen rengastustietokannalle.

1.3 Yleiskatsaus dokumenttiin

Tämä dokumentti määrittelee projektissa toteutettavan sovelluksen. Kahdessa ensimmäisessä luvussa esitellään projekti ja sen tavoitteet. Kolmas luku tulee sisältämään huomautukset vaatimusdokumenttiin tehdyistä muutoksista, mikäli sellaisia tulee.

Neljäs luku käsittelee ohjelman luokkarakennetta. Luku on jaoteltu loogisten ohjelmistokokonaisuuksien mukaan. Viidennessä luvussa perehdytään erityisesti näkyvään käyttöliittymään. Kuudennessa luvussa kerrotaan ohjelmiston suunnittelua ja toteutusta koskevat rajoitteet.

2 Sovellusalue

Ohjelmisto tarjoaa selainpohjaisen käyttöliittymän asiakkaan käytössä olevaan lintujen rengastusjärjestelmään ja se korvaa aiemman SQL*Forms sovelluskehittimellä toteutetun merkkipohjaisen käyttöliittymän. Tavoitteena on tarjota rengastustoimiston työntekijöille helppokäyttöinen ja nykyaikainen käyttöliittymä rengastusjärjestelmään.

Sovellus tuottaa dynaamisesti HTML-sivuja. Käyttäjä voi näiden sivujen avulla suorittaa hakuja tietokannasta ja ylläpitää rengastusjärjestelmän tietoja. Syötetyt tiedot tarkistetaan ennen niiden päivittämistä tietokantaan. HTML-sivuja luova koneisto ja syötteet tarkistava sovelluksen osa sijaitsevat palvelimella. WWW-Tipu ohjelmaa täytyy päästä käyttämään miltä tahansa verkkoon liitetyltä tietokoneelta, jossa on tarkoitukseen sopiva selain. Käyttäjäkunta saadaan rajattua kysymällä ohjelman alussa käyttäjältä käyttäjätunnus ja salasana.

2.1 Koko järjestelmä ja ohjelmiston rooli siinä

Rengastusjärjestelmässä ylläpidetään tietoja rengastetuista linnuista, lintujen tapaamisista, rengasjakeluista sekä näihin liittyvistä aputiedoista. Aputietoja käytetään esimerkiksi rengastusten tarkistuksiin. Ohjelmiston avulla rengastustoimiston työntekijät valvovat järjestelmässä pidettävää tietoa. He esimerkiksi ylläpitävät tietoja Suomen kunnista ja niiden maantieteellisistä koordinaateista, jotta tiedot rengastuksista ja lintuhavainnoista voitaisiin tarkistaa ja paikallistaa mahdollisimman tarkasti.

WWW-Tipu on rengastusjärjestelmän käyttöliittymä Tipukantaan. Rengastajia varten on tehty oma ohjelma, jolla he välittävät tiedot kontrolleista ja rengastamistaan linnuista järjestelmään.

3 Vaatimusmäärittely ja siihen tehdyt muutokset

Järjestelmän vaatimukset on kuvattu 12.10.2000 päivätyssä dokumentissa ”Graafinen käyttöliittymä lintujen rengastusjärjestelmään, Vaatimusdokumentti”. Mitään oleellisia muutoksia vaatimusmäärittelyyn ei ole tehty.

4 Moduulikuvaukset

Huom: Metodien kuvauksissa näkyvyysmääre on merkitty vain jos metodi on jotakin muuta kuin julkinen (`public`) tai "package private" eli ilman kirjoitettua määrettä.

4.1 Ohjelmiston toimintaperiaate

4.1.1 Ratkaisun filosofia

WWW-Tipun avulla ylläpidetään Tipukannan tietoja. Ohjelmaa tulee käyttämään rengastustoimiston henkilökunta. Käyttäjät joutuvat antamaan käyttäjtunnuksen ja salasanan järjestelmään ennen kuin he pääsevät käyttämään itse sovellusta. Valittuaan halutun näytön valikosta käyttäjä pääsee hakemaan ja päivittämään kannan tietoja. Muutamaa poikkeusta lukuunottamatta yksi lomake vastaa yhtä tietokannan taulua. Näytöllä näkyvä lomake generoidaan automaattisesti käyttäjän valintojen mukaan.

Sovelluksesta pyritään tekemään yksinkertainen ja helposti ylläpidettävä, sillä projektin päätyttyä rengastusjärjestelmän ylläpitäjän tehtäväksi jää WWW-Tipun jatkokehitys.

WWW-Tipu toteutetaan pääosin Java-kielellä, HTML-dokumenteissa käytetään apuna JavaScript-kieltä. Ratkaisuun vaikutti se, että Java on tehokas ja ilmaisuvoimainen kieli WWW:ssä toimivien ohjelmien tekemiseen. JavaScript päätettiin ottaa yhdeksi ohjelmiston toteuttavaksi kieleksi, koska sen avulla HTML-lomakkeiden käsittely helpottuu ja voidaan jakaa osa ohjelman työstä selaimessa tehtäväksi. JavaScript lisää ohjelman tehokkuutta, osa lomakkeiden tarkistuksista voidaan jo tehdä selaimessa, yksinkertaisissa virhetilanteissa tietoa ei siis tarvitse lähettää palvelimelle.

4.1.2 Toimintaperiaate

Toteutusympäristönä on Unix. WWW-Tipu käyttää valmista Oracle-tietokantaa. Tiedon lisäyksen ja päivityksen yhteydessä erillinen Fortran-kielinen Check-ohjelma tarkistaa rengastustiedot ja tallentaa hyväksytyn tiedon Tipukantaan. Kaikkien tietojen tarkistukseen ei tarvita Check-ohjelmaa, osa muutoksista voidaan tehdä suoraan tietokantaan.

Käytettävä Java servlet -moottori (Apache JServ 1.1) ja Java-kääntäjä (Sun J2EE 1.2.2) sijaitsevat TKTL:n Unix-palvelimella alkokrunni alias db.cs.helsinki.fi. Valmis WWW-Tipu ohjelma tullaan sijoittamaan ATK- keskuksen palvelimeen.

Käyttöliittymän pääsivu ladataan eläinmuseon käyttämältä Web-palvelimelta ja loput sivuista luodaan käyttäjän valintojen mukaan Unix-palvelimelle sijoitettavien servlettien avulla. HTML-lomakkeiden tiedot lähetetään servleteille HTTP-protokollan GET- ja POST-toiminnoin, tulosteet generoidaan asiakkaan selaimelle. Järjestelmän käyttö vaatii Internet Explorer- tai Netscape-selaimen version 4.0 tai uudemman.

4.2 Moduulit ja niiden väliset suhteet

WWW-Tipu koostuu neljästä päämoduulista, jotka ovat:

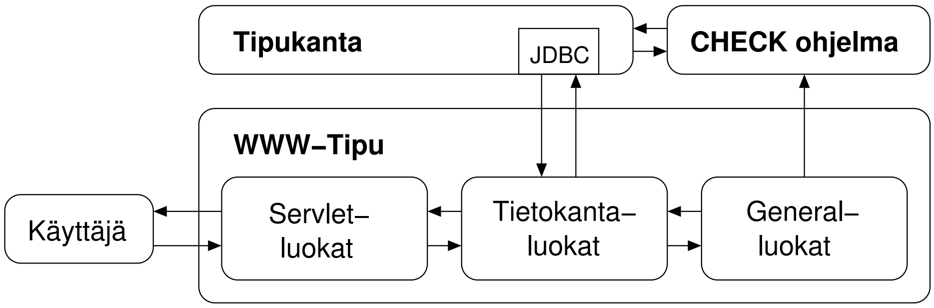
Java Servlet -luokat Nämä luokat hoitavat kommunikoinnin käyttäjän kanssa HTTP-protokollan mukaisesti.

JDBC-tietokanta-yhteys Hoitaa yhteyden relaatiotietokantaan, jonne tiedot rengastuksista on talletettu.

General-luokat Tarvitaan muiden luokkien asetuksia ja Check-ohjelmaa varten.

Check-ohjelma Fortran-kielinen ohjelma, joka hoitaa käyttäjän antaman datan validoinnin ennen kuin se lisätään tietokantaan.

Moduulien välinen kommunikaatio on kuvattu kuvassa 1. Käyttäjän syöttämä informaatio lähetetään Servlet-luokille, jotka sitten jäsentävät sen ja välittävät operaation tietokantayhteysluokalle. Suoritettava toimenpide voi olla haku-, lisäys-, muokkaus- tai poisto- operaatio.

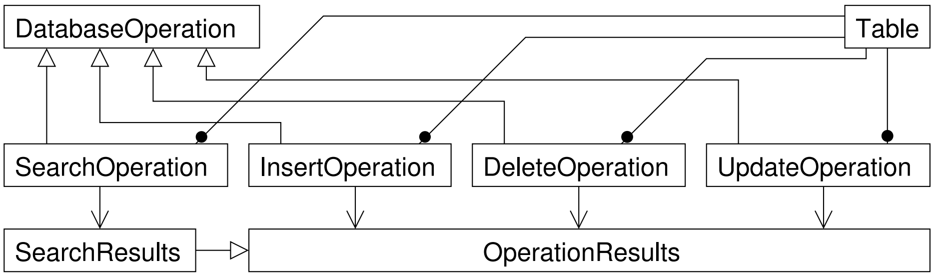


Kuva 1: Yleiskuva ohjelmiston moduuleista ja niiden välisestä kommunikaatiosta.

Rengastustapahtuman tietoja lisättäessä tietokantayhteysluokka kutsuu Check-ohjelmaa tietojen muodollista tarkistamista varten ennen niiden lisäämistä tietokantaan. Operaation jälkeen Servlet-luokat lukevat tietokannasta operaation tulokset ja palauttavat ne käyttäjälle.

4.3 Tietokantayhteys

JDBC-yhteys tietokantaan hoidetaan `tipu.db` pakkauksen luokkien avulla. Yhteyksiä jaetaan yhteysvarannon (connection pool) avulla. Täten mahdollisesti monia muitakin käyttäjiä palvelevaa tietokantaa ei missään vaiheessa ruuhkauteta tiettyä rajaa enemmän.



Kuva 2: Tietokantaoperaatioiden luokkarakenne ja tulostyypit.

Tietokantaan voi kohdistua neljä erilaista operaatiota: haku, lisäys, poisto tai muutos. Kuvasta 2 nähdään, että nämä operaatiot on toteutettu luokissa `SearchOperation`, `InsertOperation`, `DeleteOperation` ja `UpdateOperation`. Nämä kaikki periytyvät yhteisestä abstraktista kantaluokasta `DatabaseOperation`, joka käytännössä tarjoaa tuen `ConnectionPool`:in dynaamiselle yhteysjaolle.

Operaatiot palauttavat kutsujalleen lisäys-, muutos- ja poistotapauksissa totuusarvon, joka indikoi halutun operaation onnistumista. Haku palauttaa erillisen `SearchResults`-objektin, jolla hakutuloksia voidaan tarkastella lähemmin.

Kuva 3 antaa käyttöesimerkin poisto-operaatiosta (ks. 4.3.10). Esimerkistä ilmenee yhteyden varaus, operaation alustus ja suoritus, sekä tulosten tarkastelu.

4.3.1 Yhtäaikaisuus

Käyttöliittymän toteutusalustan (HTTP) yhteydettömästä luonteesta johtuen ei kannasta voida varata resursseja yhden pyynnön käsittelyä pidemmäksi ajaksi. Palvelin ei käytännössä voi varmasti tietää milloin jokin käyttäjä on lopettanut kannan käytön, esim. vain sulkemalla selainikkunansa.

Tästä syystä mm. edellisen käyttöliittymän ominaisuus, joka lukitsi esillä olevan tietueen muokkausta varten, on nyt mahdoton toteuttaa. Seurauksena saattaa ol-

Request

```
Table target = new Table("KUNNAT");

Map keys = new HashMap();
keys.put("KUNIMI", "HELSINKI");

Connection con = ConnectionPool.requestConnection();

DeleteOperation delete = new DeleteOperation(target);
boolean success = delete.executeDelete(con, keys);

if (success) {
    response.println("Success");
}
else {
     response.println("Failure");
}

ConnectionPool.releaseConnection(con);
```

Response

Kuva 3: Yksinkertaistettu esimerkki tietokantaluokkien käytöstä servletin suorittamassa poisto-operaatiossa.

la esim. peräkkäisten päivitysten ongelma: Kun kaksi käyttäjää muokkaa samaa tietuetta ja kumpikin tallettaa sen yhtäaikaa.

Tapahtumien onnistuminen riippuu tietokannan transaktioiden eristäytyneisyysasteesta ja toimintamallista tilanteissa, joissa yritetään operoida lukittua tietuetta. WWW-Tipu asettaa jokaiselle toimittamalleen tietokantaoperaatiolle sen vaatiman eristäytyneisyysasteen ja toimii operaatioiden antaman palautteen perusteella. Mikäli esim. päivitys epäonnistuu, tästä ilmoitetaan käyttäjälle, ja tarjotaan mahdollisuus yrittää toimintoa uudestaan.

4.3.2 Luokka `tipu.db.ConnectionPool`

Yhteysvaranto (connection pool) sisältää kaikki käytettävät JDBC- tietokantayhteydet. Servlet pyytää varannosta yhteyden käyttöönsä jokaisen HTTP-pyynnön alussa ja suorittaa kaikki kyseisen pyynnön aikana tehtävät tietokantaoperaat annettua yhteyttä pitkin.

Yhteysvarantoa käytetään, jotta kaikille käyttäjille voitaisiin taata tasaisesti tietty palvelun taso. Tiettyyn maksimiin rajoitettu yhtäaikaisten tietokantaoperaatioiden määrä takaa seuraavaa:

- Suorituksessa olevat operaatiot saavat jokainen käyttä vähintään tietyn minimimäärän tietokannan resursseista.
- Uusille asiakkaille voidaan välittömästi ilmoittaa, mikäli palvelu on niin ruuhkainen, ettei ole kenenkään edun mukaista yrittää suorittaa enää uusia operaatioita juuri kyseisellä hetkellä.

Staattinen rajapinta

Tietokantaoperaation suoritukseen tarvittava yhteys pyydetään metodilla `requestConnection()` ja se täytyy myöhemmin vapauttaa metodilla `releaseConnection()`. Käyttäjä (servlet) näkee vain nämä kaksi metodia, ja käyttää niitä kuvan 3 osoittamalla tavalla.

`Connection requestConnection()`

Tämä metodi pyytää yhteysvarannosta yhteyttä. Suoritus jää tarvittaessa odottamaan, mikäli yhteyttä ei välittömästi saada.

`void releaseConnection(Connection con)`

Tämä metodi vapauttaa aiemmin varatun yhteyden. Näin tulisi tehdä viimeistään silloin kun kyseisen pyynnön vastaus lähetetään. Yhteydelle voi silti jättää esim. kursoreita auki, sillä itse yhteys säilyy avoimena, vaikka se luovutettaisiinkin toisten operaatioiden käyttöön.

Näin saattaa käydä esim. silloin, kun käyttäjä selaa jonkin taulun tietueita (käyttää kursoria) tehden niihin välillä muutoksia (suorittaa päivitysoperaatioita). Haku-operaatiota suoritetaan aina samalla yhteydellä millä se on aikanaan käynnistettykin (`java.sql.ResultSet` on sidottu sen suorittaneeseen yhteyteen), mutta päivitys voi saada käytettäväkseen minkä tahansa yhteysvarannon yhteyden.

`private ConnectionPool getInstance()`

Tämä metodi palauttaa itsenäisen `ConnectionPool` -instanssin. Tätä metodia kutsuu ainoastaan `requestConnection()` ensimmäisellä kutsukerrallaan.

Instanssin rajapinta

Tämän luokan instanssi on rajoitettu palvelemaan ainoastaan luokan staattista rajapintaa. Siksi luokan konstruktori on näkyvyydeltään `private`. Toteutus on kuitenkin instanssiin pohjautuva, sillä luokka vaatii verrattaen laajan alustusoperaation (ks. konstruktori `ConnectionPool()`), joka on kätevintä toteuttaa juuri konstruktorissa, staattisten alustusmetodien sijaan.

`private ConnectionPool()`

Konstruktori suorittaa seuraavat alustusoperaatiot:

- Luodaan joukot (`java.util.Set`) käytetyille ja käyttämättömille yhteyksille.
- Haetaan tietokannan yhteystiedot luokan `tipu.general.TipuProperties` avulla ja instantoidaan JDBC-ajuri (tekniikka dokumentoitu Sunin JDBC 2.0 spesifikaatiossa [6]).

- Muodostetaan ensimmäinen käyttövalmis yhteys palvelemaan ensimmäistä asiakasta.

`private Connection getConnection()`

Tämä metodi hankkii kutsujalleen yhteyden jollakin seuraavista keinoista:

- Mikäli vapaiden yhteyksien joukosta löytyy toimiva yhteys, otetaan se. Jos yhteys on kuitenkin ollut niin kauan käyttämättömänä, että se on jo ajurin puolesta suljettu, luodaan sen tilalle uusi yhteys, ja *yritetään* yhteyden varausta uudelleen.

  Tässä vaiheessa alkuperäinen yhteyden kysyjä kilpailee tasavertaisesti mahdollisesti muidenkin yhteyttä pyytämään tulleiden asiakkaiden kanssa vapaista yhteyksistä.

- Jos vapaita yhteyksiä ei ole, eikä yhteyksiä ei ole vielä muodostettu suurinta sallittua määrää, luodaan uusi yhteys ja kilpailutetaan se em. tavalla. Mikäli yhteyksiä ei voida enää luoda, jäädään odottamaan, että jokin käytössä olevista yhteyksistä ilmoittaa vapautuneensa.

`private Connection makeNewConnection()`

Tämä metodi pyytää `java.sql.DriverManager` -luokalta uuden yhteyden. Epäonnistunut pyyntö palauttaa arvon `null`.

`private void makeBackgroundConnection()`

Tämä metodi käynnistää uuden säikeen muodostamaan yhteyttä. Yhteys muodostetaan säikeen avulla, koska yhteyden neuvottelu voi kestää useamman sekunnin. Kun se suoritetaan taustalla, voidaan palvella muita yhteyttä pyytämään tai vapauttamaan tulleita asiakkaita rinnalla.

Kaikki yhteyttä odottavat asiakkaat herätetään kilpailemaan yhteydestä heti, kun uusi yhteys saadaan käyttöön.

`private void free(Connection con)`

Tämä metodi vapauttaa tähän yhteysvarantoon kuuluvan yhteyden. Yhteys siirtyy välittömästi vapaiden yhteyksien joukkoon ja mahdollisille odottajille ilmoitetaan tästä.

`private void closeConnections(Set)`

Tämä metodi sulkee kaikki annetussa yhteysjoukossa olevat yhteydet. Metodia käyttää lähinnä vain `closeAllConnections()`.

`private void closeAllConnections()`

Tämä metodi sulkee kaikki avoimet yhteydet, niin käyttämättömät kuin käytössä olevatkin. Metodia kutsutaan `ConnectionPool`-instanssin tuhoamisen yhteydessä `finalize()`-metodin kautta.

`protected void finalize()`

Javan garbage collection kutsuu tätä metodia, joka puolestaan kutsuu `close-AllConnections()` -metodia sulkeakseen kaikki auki olevat yhteydet. (Vaihe ei sinänsä ole tarpeellinen sillä yhteydet osaavat myös itse sulkea itsensä siististi kun niiden `finalize()` -metodia kutsutaan.)

4.3.3 Luokka `tipu.db.Table`

`Table` sisältää jonkin tietokannan taulun ns. metadatan, eli (1) attribuutit, (2) pääavaimet ja (3) attribuuttien kommentit. Jokaisella servletillä on yksi tai useampi kohdetaulu, johon kaikki servletiltä pyydetyt operaatiot kohdistuvat. Servlet hakee käynnistyessään jokaisen kohdetaulunsa `Table` -objektin, ja säilyttää niitä jäsenmuuttujissaan elinkaarensa ajan.

Vain oikeat tiedot sisältävän `Table` -instanssin avulla tietokantaoperaatiot voidaan kohdistaa johonkin tiettyyn tauluun, koska tarvittavat SQL-lauseet osataan rakentaa ainoastaan kohdetaulun attribuutti- ja avaintietojen perusteella.

Instanssin rajapinta

Luokan `Table` instanssi tarjoaa käyttäjälleen metodit attribuutti- ja kommenttitietojen hakemiseen. Avaimet ja attribuutit palautetaan järjestetyissä joukoissa, jotta tietokantaoperaatioissa osattaisiin myöhemmin sijoittaa parametrit valmisteltuihin lausekkeisiin (`java.sql.PreparedStatement`) oikeille paikoille.

`private Table(String tableName, SortedSet keys, SortedSet attributes, Map comments)`

Konstruktori sijoittaa parametreina saamansa arvot ja arvojoukot jäsenmuuttujiinsa.

`String getName()`

Tämä metodi palauttaa kohteena olevan taulun nimen.

`SortedSet getAttributes()`

Tämä metodi palauttaa taulun attribuuttien (eli sarakkeiden) nimet joukkona `String` -objekteja. Joukko on määrittelynsä (ks. [5] luokka `java.util.SortedSet`) mukaisesti aina samassa järjestyksessä. Käytännössä tämä on `String` -objekteilla nouseva aakkosjärjestys.

`SortedSet getKeys()`

Tämä metodi palauttaa taulun pääavainten nimet joukkona `String` -objekteja. Mikäli joukossa on useita attribuutteja, ne ovat joukon sisällä järjestettynä samoin kuin kaikki attribuutit käsittävä joukko.

Huom: Tämä ei ole avainten taulumäärittelyn mukainen järjestys, vaan ainoastaan se järjestys, jossa attribuutit tullaan sijoittamaan tietokantaoperaatiossa luotavaan SQL-lauseeseen.

`String getComment(String attribute)`

Tämä metodi palauttaa pyydetyn attribuutin kommentin, mikäli sellainen on tiedossa. Muutoin palautetaan tyhjä merkkijono. Kommentteja käytetään pika-avusteina HTML-sivuilla näkyvien kenttien yhteydessä.

Staattinen rajapinta

Luokan `Table` instansseja pyydetään staattisen rajapinnan avulla.

`Table getTable(String tableName)`

Hakee resurssitiedostosta taulun *`tableName`* tiedot ja luo niistä käyttövalmiin `Table` -instanssin.

4.3.4 Luokka `tipu.db.DatabaseOperation`

`DatabaseOperation` on abstrakti yleistys kaikista mahdollisista servletien tietokantaan kohdistamista operaatioista.

Jokaisella operaatiolla oletetaan olevan yksi tiettyyn tauluun kohdistuva valmisteltu lauseke (`java.sql.PreparedStatement`), jollaisen instanssia `DatabaseOperation` säilyttää jokaista ko. operaation suoritukseen tarjottua tietokantayhteyttä kohti. Näin täytyy tehdä, sillä valmistellut lausekkeet ovat sidottuja siihen yhteyteen, jolla valmistelu on alkujaan suoritettu.

Instanssin rajapinta

Jokainen tietokantaoperaatio on oltava valmisteltavissa siten, ettei joka kerta kyseistä operaatiota tiettyyn tauluun kohdistettaessa tarvitse kyselylauseketta kääntää uudestaan. Siksi jokaisen tietokantaoperaation on toteutettava tietty tietokantaoperaatioiden hallinnalle yhteinen rajapinta.

`DatabaseOperation(Table table)`

Konstruktori luo geneerisen tietokantaoperaation, joka kohdistuu tauluun *`table`*.

`PreparedStatement getPreparedStatement(Connection)`

Tämä metodi palauttaa annetulle yhteydelle valmistellun tätä operaatiota vastaavan lausekkeen. Mikäli valmistelua ei juuri tälle yhteydelle vielä ole tehty, se tehdään metodia kutsuttaessa, ja talletetaan tämän operaation instanssiin käytettäväksi juuri tälle yhteydelle tämän operaation tulevia suorituksia varten.

`int getResultSetType()`

Tämä metodi palauttaa operaation vaatiman/salliman vastausjoukon kursorityypin, jonka tulee olla on yksi `java.sql.ResultSet` luokan arvoista `TYPE_FORWARD_ONLY`, `TYPE_SCROLL_INSENSITIVE` tai `TYPE_SCROLL_SENSITIVE`. Oletuksena palautetaan arvo `TYPE_FORWARD_ONLY`.

`int getResultSetConcurrency()`

Tämä metodi palauttaa operaation vaatiman/salliman tietueiden eristyneisyysasteen, jonka tulee olla toinen `java.sql.ResultSet` luokan arvoista `CONCUR_READ_ONLY` tai `CONCUR_UPDATABLE`. Oletuksena palautetaan arvo `CONCUR_READ_ONLY`.

Abstrakti rajapinta

Ainoa asia mitä geneerinen tietokantaoperaatio ei voi määritellä on SQL-lause, joka määrää mitä kyseinen operaatio käytännössä tekee. Siksi jokaisen erikoistuneen operaation on toteutettava luokan `DatabaseOperation` abstrakti metodi `getSQLString()`.

`String getSQLString()`

Palauttaa toteuttavan operaation SQL-kielisen lausekkeen, johon on objektia luotaessa annetun kohdetaulun avainjoukon avulla luotu valmis parametrilista.

4.3.5 Luokka `OperationResults`

Jokainen tietokantaoperaatio päättyy tulokseen, joka on luettavissa operaation `execute`-metodin palauttamasta `OperationResults` -tyyppisestä objektista. Perustoteutus tarjoaa rajapinnan totuusarvoisen tuloksen ja virheilmoituksen lukemiseen. Monimutkaisempaa tulosten raportointia varten voidaan tästä luokasta periyttää uusi tulosobjekti. Näin on tehty esim. hakutulosten kanssa (ks. 4.3.7).

Instanssin rajapinta

Koska jokainen operaatio voi vähintäänkin onnistua tai epäonnistua, tarjoaa jokainen tulosobjekti metodit tämän tilan selvittämiseen. Lisäksi tarjotaan metodit tilan asettamiseen. Näitä metodeja käyttävät vain `tipu.db.DatabaseOperation` -luokasta periytyvät luokat, joten niiden näkyvyys on "package private" (eli ei merkittyä määrettä).

`boolean succeeded()`

Tämän metodin avulla palvelinsovelma voi nopeasti päätellä ryhdytäänkö virheenkäsittelyoperaatioihin, vai jatketaanko normaalin vastauksen tuottamisella. Metodi palauttaa siis `true`, mikäli virheitä ei tapahtunut.

`String getErrorMessage()`

Mikäli suoritettu operaatio epäonnistui (`succeeded()` palauttaa epätoden), tällä metodilla voidaan kysyä käyttäjälle ilmoitettava virheilmoitus. Sovelma ei tämän tekstin perusteella voi suorittaa mitään korjaavia toimenpiteitä. Teksti on tarkoitettu ainoastaan virheen raportointiin.

Mikäli virheilmoitusta ei ole tietokantaoperaatiossa annettu, palauttaa tämä metodi tyhjän merkkijonon.

`void setSuccess(boolean state)`

Tällä metodilla asetetaan tuloksen tila joko onnistuneeksi (`true`) tai epäonnistuneeksi (`false`).

`void setErrorMessage(String message)`

Tällä metodilla asetetaan tuloksen virheilmoitus.

4.3.6 Luokka `tipu.db.SearchOperation`

`SearchOperation` suorittaa `SELECT`-lauseella haun tietokannan tauluun annetuilla attribuuttimaskeilla. Kaikki vertailut tehdään `LIKE`-operaattorein, joten käyttäjät voivat itse syöttää jokerimerkkejä hakukenttiin. Tulokset palautetaan erillisessä `SearchResults`-objektissa.

Instanssin rajapinta

Hakuoperaation rajapinta on pääosin sama kuin muidenkin tietokantaoperaatioiden. Haku suoritetaan metodilla `executeSearch()` antamalla parametriksi kartta hakuarvoista. Tuloksena käyttäjä saa erillisen tulosobjektin `SearchResults`.

`SearchOperation(Table table)`

Konstruktori luo uuden, tauluun *`table`* kohdistuvan hakuoperaation.

`SearchResults executeSearch(Connection con, Map values)`

Tämä metodi suorittaa hakuoperaation kohdetauluun annetulla yhteydellä *`con`* hakuarvoilla *`values`*. Kutsu palauttaa `SearchResults` -instanssin (ks. 4.3.7), joka sisältää haun palauttaman `java.sql.ResultSet` -kursorin, ja jonka avulla hakutuloksia voidaan tutkia.

`private void setParameters(PreparedStatement stmt, Map values)`

Tämä metodi sijoittaa annetut hakuarvot kyselylausekkeen parametreiksi. Tyhjät ja puuttuvat arvot korvatan SQL:n `LIKE` -operaattorin jokerimerkillä '`%`', jolloin nämä attribuutit sallivat minkä arvon tahansa.

`protected String getSQLString()`

Tämä metodi palauttaa hakulausekkeen muotoa:

```
SELECT attr1, attr2, ..., attrN FROM taulu
WHERE attr1 LIKE '?' AND attr2 LIKE '?' AND
... AND attrN LIKE '?'
```

Kentät `attr?` ovat kohdetaulun attribuutteja. Kysymysmerkkien paikalle `PreparedStatement` -objektiin sijoitetaan operaation suoritukseen käytettävät parametrit.

`protected int getResultSetType()`

Palauttaa vakion `java.sql.ResultSet.TYPE_SCROLL_SENSITIVE`, sillä tulosjoukkoa on voitava selata edestakaisin, ja kantaan tehtyjen muutosten tulee näkyä hakutuloksissa saman tien.

Esim: käyttäjä hakee tietyn joukon taulun rivejä, muuttaa niistä yhtä, siirtyy seuraavaan ja siitä heti takaisin edelliseen. Juuri tehtyjen muutosten tulee tällöin ehdottomasti näkyä ruudulla.

`protected int getResultSetConcurrency()`

Palauttaa vakion `java.sql.ResultSet.CONCUR_UPDATABLE`, sillä hakutulosten selaus voi kestää kauan ja toisten käyttäjien on tällä välin päästävä muuttamaan kyseisen taulun arvoja. Tästä syystä kursori ei saa lukita rivejä itselleen, vaan niiden muuttaminen on sallittava.

4.3.7 Luokka `tipu.db.SearchResults`

Luokka `SearchResults` periytyy luokasta `OperationResults`. Lisäksi se sisältää hakuoperaation palauttaman `java.sql.ResultSet` -objektin, ja mahdollistaa sen palauttamien tietojen selaamisen käytännöllisesti (käyttävän servletin kannalta).

Haun suorittava servlet säilyttää kullekkin sessiolle viimeisintä hakutulosta ja lukee sieltä arvoja aina kun käyttäjä liikkuu tuloslistassa eteen- tai taaksepäin.

Instanssin rajapinta

Hakuoperaatiota kutsunut servlet pääsee käsiksi hakutuloksiin luokan `SearchResults` instanssin rajapinnan kautta. Suurin osa seuraavista metodeista heijastaa kursorin toiminnallisuutta, joten hakujoukossa voi liikkua samoin askelin kuin käyttäjä valitsee näytöltä hakutuloksia: seuraava, edellinen tai satunnainen tietue.

Hakuoperaatiolle ominaisten metodien lisäksi `SearchResults` sisältää myös luokan `OperationResults` määrittelemät yleiset operaation tilaa käsittelevät operaatiot.

`protected SearchResults(ResultSet rs)`

Konstruktori luo instanssin, joka käsittelee annettua vastausjoukkoa.

`int getRow()`

Tämä metodi palauttaa sen rivin numeron, jolla kursori tällä hetkellä on. Arvo on väliltä $1..n$, missä n on tulosjoukon rivien määrä.

`SortedMap getCurrentRow()`

Tämä metodi palauttaa vastausjoukon senhetkisen rivin järjestettynä (attribuutti $\rightarrow$ arvo) karttana.

`SortedMap getNextRow()`

Tämä metodi palauttaa järjestyksessä nykyistä kohtaa seuraavan rivin. Kursoria siirretään yksi rivi eteenpäin. Metodi palauttaa arvon `null`, mikäli kursori on viimeisellä rivillä.

`SortedMap getPreviousRow()`

Tämä metodi palauttaa järjestyksessä nykyistä kohtaa edeltävän rivin. Kursoria siirretään yksi rivi taaksepäin. Metodi palauttaa arvon `null`, mikäli kursori on ensimmäisellä rivillä.

`SortedMap getAbsoluteRow(int row)`

Tämä metodi palauttaa käyttäjän määrittelemän absoluuttisen rivin. Rivien numerointi alkaa yhdestä.

`List getRowList()`

Tämä metodi palauttaa kaikki hakutuloksen rivit `java.util.List` -rajapinnan toteuttavana objektina. Lista koostuu `java.sql.SortedMap` -objekteista joista jokainen kuvaa yhden taulukon rivin.

4.3.8 Luokka `tipu.db.InsertOperation`

Lisäysoperaatiolla voidaan lisätä yksi rivi määrättyyn tietokannan tauluun. Lisäys onnistuu, mikäli taulussa ei vielä ole samoilla avaimilla varustettua tietuetta, taulun eheys- ja avainrajoitteet toteutuvat eikä tiedonsiirtovirheitä tapahdu.

Instanssin rajapinta

Keskustelu lisäysoperaation (kuten muidenkin operaatioiden) kanssa on hyvin suoraviivaista. Ensin operaatio konstruoidaan antamalla parametriksi kohdetaulu. Tämän jälkeen koostetaan arvot kartaksi ja kutsutaan niillä metodia `executeInsert()`, joka suorittaa itse operaation.

`InsertOperation(Table table)`

Konstruktori luo uuden, tauluun *`table`* kohdistuvan lisäysoperaation.

`OperationResults executeInsert(Connection con, Map values)`

Tämä metodi lisää rivin kohdetauluun arvoilla *`values`*. Ko. arvokartasta tulee löytyä arvot ainakin kaikille pääavaimille sekä muille “`NOT NULL`” -attribuuteille.

`private void setParameters(PreparedStatement stmt, Map values)`

Tämä metodi kopioi annetut arvot SQL-lausekkeeseen. Tätä kutsutaan vain metodin `executeInsert()` sisältä.

`protected String getSQLString()`

Tämä metodi palauttaa SQL-lauseen muotoa:

```
INSERT INTO taulu (attr1, attr2, ..., attrN)
VALUES ('?', '?', ..., '?')
```

Kentät *`attr?`* ovat kohdetaulun attribuutteja.

4.3.9 Luokka `tipu.db.UpdateOperation`

Päivitysoperaatio muuttaa jonkin jo olemassa olevan rivin attribuuttien arvoja. Operaatio onnistuu, mikäli kohdetaulusta löytyy annetun ROWID:n osoittama rivi, eikä tiedonsiirtovirheitä tapahdu.

Operaatiolle täytyy antaa parametrina muutettavan tietueen ROWID, jotta ylipäänsä voidaan tietää mitä taulun riviä tulee muuttaa.

Instanssin rajapinta

Muutosoperaatiota kutsutaan metodilla `executeUpdate`, joka saa parametrikseen kyseisen tietueen ROWID:n sekä uudet arvot. Uusilla arvoilla tarkoitetaan tässä kaikkien attribuuttien tulevia arvoja. Käytännössä päivitykset tehdään aina kaikille attribuuteille kerralla, sillä arvot haetaan aina servletille lähetetystä lomakkeesta, jossa jokainen kenttä on määritelty.

`UpdateOperation(Table table)`

Konstruktori luo uuden, tauluun *`table`* kohdistuvan päivitysoperaation.

`OperationResults executeUpdate(Connection con, Map values, String rowId)`

Tämä metodi suorittaa päivitysoperaation taulun riville *`rowId`*, muuttaen tämän rivin attribuutit kartan *`values`* mukaisesti.

`private void setParameters(PreparedStatement stmt, Map values, String rowId)`

Tämä metodi kopioi uudet attribuuttien arvot sekä ROWID:n kyselylausekkeeseen. Annetut arvot korvaavat *kaikki* vanhat arvot. Mikäli arvokartasta *`values`* ei jotakin arvoa löydy, ko. kohtaan asetetaan arvo `NULL`. Eheysrajoitteen `NOT NULL` sisältävien kenttien kohdalla tästä aiheutuu väistämättä poikkeus ja toiminnon epäonnistuminen.

Käytännössä kaikkien kenttien päivityksestä ei kuitenkaan synny ongelmaa, sillä päivitysoperaatiota suoritettaessa servletin kutsuobjekti `javax.servlet.HttpServletRequest` sisältää tietueen kaikkien attribuuttien arvot - myös vanhat arvot niistä kentistä joita käyttäjä ei ole muuttanut.

`protected String getSQLString()`

Tämä metodi palauttaa SQL-lauseen muotoa:

```
UPDATE taulu
SET attr1 = '?', attr2 = '?', ..., attrN = '?'
WHERE ROWID = '?'
```

Kentät `attr?` ovat kohdetaulun attribuutteja.

4.3.10 Luokka `tipu.db.DeleteOperation`

Poisto-operaatio tuhoaa yhden rivin kohdetaulusta. Operaatio onnistuu, mikäli taulusta löytyy annettu rivi eikä tiedonsiirtovirheitä tapahdu.

Instanssin rajapinta

Poisto-operaatio käynnistetään kutsumalla metodia `executeDelete()`, joka saa parametrinaan poistettavan rivin ROWID:n.

`DeleteOperation(Table table)`

Konstruktori luo uuden, tauluun *`table`* kohdistuvan poisto-operaation.

`OperationResults executeDelete(Connection con, String rowId)`

Tämä metodi tuhoaa kohdetaulusta rivin annetulla ROWID:llä.

`private void setParameters(PreparedStatement stmt, String rowId)`

Tämä metodi kopioi ROWID:n arvon SQL-lausekkeeseen.

`protected String getSQLString()`

Tämä metodi palauttaa SQL-lauseen muotoa:

```
DELETE FROM taulu
WHERE ROWID = '?'
```

4.4 Servlet-luokat

Lomakkeiden toiminnallisuus toteutetaan servlet-luokilla, joita on yksi kutakin lomakesivua (esim. kunnat) kohden. Lisäksi lomakkeilta avattavia popup-valikoita (esim. lista rengastajien nimistä) varten on oma Popup-luokkansa, joka on yhteiskäytössä eri näyttöjen kesken. Pääpiirteittäin luokat ovat toistensa kaltaisia, erot johtuvat lähinnä eri lomakesivujen sisältämistä kentistä.

Servlet-luokan suoritus käynnistyy, kun servletiä kutsutaan WWW-selaimella ensimmäistä kertaa servlet-palvelimen käynnistyttyä. Tästä eteenpäin servlet on aktiivisena palvelimen muistissa niin kauan, kunnes palvelin suljetaan. Servlet käyttää säikeitä palvellakseen useita yhtäaikaisia pyyntöjä ja voi myös sisältää staattista, kaikkien pyyntöjen käytössä olevaa informaatiota.

Servlet vastaanottaa informaatiota WWW-selaimelta HTTP-protokollan GET- ja POST-toiminnoilla ja voi istunnon lopuksi palauttaa selaimelle generoimansa WWW-sivun.

Lomakesivujen toiminnallisuuden toteuttavat servlet-luokat muodostavat WWW-Tipun ytimen. Servletit kutsuvat muita `tipu.*` pakkauksien luokkia mm. tietokantaa käsitellessään ja properties-tiedostoja lukiessaan, sekä lisäksi ulkoisia kirjastoja esim. templatetiedostojen tulostamiseksi.

Poistettaessa tietuetta taulusta, jossa on viite-eheysrajoitteita, poistetaan samalla kaikki tietueeseen liittyvät rivit. Ennen varsinaista poistoa servlet hakee kannasta jokaisen poistettavaan tietueeseen liittyvän tietokantarivin ja kysyy käyttäjältä vahvistusta näiden poistoon. Monivaiheiset poistot on määritelty jokaista näyttöä kohti erikseen siihen liittyvässä servletissä.

4.4.1 Luokat `tipu.servlet.*`

Instanssin rajapinta

Kutsut Servlet-luokkiin tulevat servlet-palvelimelta HTTP-pyyntöjen muodossa. Tarkka kutsujärjestys on määritelty Sunin Servlet -spesifikaatiossa [9].

`void init(ServletConfig config)`

Huolehtii toiminnoista, jotka tulee suorittaa vain servletin käynnistymisen yhteydessä, esim. määrittelytiedostojen ja muuttumattomien tietojen (kuntanimet) lukeminen.

`doPost (HttpServletRequest request, HttpServletResponse response)`

Servletin päämetodi, joka saa parametrina WWW-selaimen post-muodossa lähettämän informaation (lomakkeen sisältö) ja HttpServletResponse- muotoisen vastauksen, johon servletin lähettämä vastaus generoidaan.

`getValues (HttpServletRequest request, Hashtable values)`

Hakee HttpServletRequestilta kaikki arvojoukot (avain, arvo) ja tallentaa ne parametrina annettuun Hashtable-tietueeseen.

`putValues (Hashtable values, SimpleHash modelRoot)`

Muutetaan Hashtable-muotoinen tietue templatekirjaston ymmärtämään muotoon ennen WWW-sivun muodostamista. Arvojoukkoja ei voida säilyttää pelkästään SimpleHash-muodossa, koska tällöin sisällön käsittely olisi mahdotonta.

4.5 Templatetiedostot

WWW-Tipu käyttää lomake-sivujen laatimiseen ilmaista (GPL) FreeMarker -templatekirjastoa, jolla sivujen ulkoasun määrittely voidaan eriyttää muusta toiminnallisuudesta. FreeMarkerin dokumentaatio löytyy seuraavista osoitteista:

- Manuaali [8]: http://freemarker.sourceforge.net/docs/manual/
- Javadoc [2]: http://freemarker.sourceforge.net/docs/

4.6 Apuluokat

4.6.1 Luokka `tipu.general.TipuProperties`

Luokka `tipu.general.TipuProperties` sisältää ohjelmiston WWW-Tipu vaatimat asetukset, mm. käytettävän JDBC-yhteyden parametrit, Check-ohjelman kutsutavan, ym.

Luokka lataa asetukset tiedostosta `etc/www-tipu.config` silloin kun niitä ensimmäisen kerran tarvitaan.

Staattinen Rajapinta

Rajapinta koostuu pyyntömetodeista, joilla voi kysyä haluamansa asetusavaimen arvon. Arvo voidaan tulkita ja palauttaa merkkijonona, numerona tai totuusarvona.

`String get(String key, String defaultValue)`

Palauttaa arvonaan asetusparametrin *`key`* arvon. Jos parametrilla ei ole arvoa, palauttaa arvon *`defaultValue`*.

`String get(String key)`

Muuten kuin ylläoleva, mutta palauttaa `null`-arvon, jos parametrilla ei ole arvoa.

`int getInt(String key, int defaultValue)`

Palauttaa arvonaan asetusparametrin *`key`* arvon kokonaislukuna, tai -1 jos arvoa ei voitu tulkita kokonaisluvuksi. Jos parametrilla ei ole arvoa, palauttaa arvon *`defaultValue`*.

`int getInt(String key)`

Muuten kuin ylläoleva, mutta palauttaa arvon -1, jos parametrilla ei ole arvoa.

`boolean getBoolean(String key, boolean defaultValue)`

Palauttaa arvonaan asetusparametrin *`key`* arvon totuusarvona, tai `false` jos arvoa ei voitu tulkita totuusarvoksi. Jos parametrilla ei ole arvoa, palauttaa arvon *`defaultValue`*.

`boolean getBoolean(String key)`

Muuten kuin ylläoleva, mutta palauttaa arvon `false` jos parametrilla ei ole arvoa.

4.6.2 Luokka `tipu.general.Exec`

Luokkaa `tipu.general.Exec` käytetään ohjelmiston WWW-Tipu tarvitseman Check-ohjelman ajamiseen. Tapa jolla ohjelma ajetaan on spesifioitu tiedostossa `etc/www-tipu.config`.

Staattinen rajapinta

Rajapinnan muodostavat kaksi variaatiota samasta metodista, joiden kummankin tehtävänä on ajaa annettu Check-ohjelma tietyin parametrein.

`boolean exec(command, session, screen, parameter)`

Suorittaa Check-ohjelman, jonka nimi on annettu parametrina *`command`*. Muut parametrit ovat istunnon, näytön ja näytöstä riippuvan parametrin (esim. diarionumeron) arvot. Palauttaa arvonaan `true` jos ohjelman suoritus onnistui, muuten `false`. Kaikki parametrit ovat tyyppiä `String`.

`boolean exec(session, screen, parameter)`

Muuten kuin ylläoleva, mutta käyttää luokkaa `TipuProperties` hakemaan Check-ohjelman nimen asetustiedostosta. Oletusarvona on `CHECK`.

5 Käyttöliittymä

5.1 Yleistä

WWW-Tipua käytetään selaimen kautta ja se koostuu dynaamisesti luoduista sivuista jotka sisältävät ainoastaan HTMLää ja JavaScriptiä. Koska eri selaimet ovat toteuttaneet joitakin asioita eri tavoin, tämä voi johtaa siihen että eri selainten välillä käyttöliittymässä esintyy pieniä eroavaisuuksia. Käyttöliittymän rakenne on sama kuin vanhassa järjestelmässä.

Sivuilla tulisi pystyä navigoimaan ilman selaimen omia navigointitoimintoja kuten esim. “Takaisin” (Back) ja “Eteen” (Forward). Mahdollisuus näiden käyttöön tulee kuitenkin ottaa huomioon toteutuksessa. Nykyisestä selaintekniikasta johtuen ohjelmassa ei voi käyttää vanhasta järjestelmästä tuttuja näppäinyhdistelmiä tomintojen suorittamiseen. Käyttäjä suorittaa halutut toiminnot hiiren avulla.

Käyttäjä käyttää järjestelmää usein ja paljon, joten hänen toimintojaan ei tarvitse ohjata vaan työn tehokkuus on tärkeää.

Ohjelmassa on mahdollista päivittää saamaa kenttää peräkkäin eri arvoilla. Tälaisessa tapauksessa yhden käyttäjän tekemät muutokset tietueeseen korvataan toisen käyttäjän tallentamilla tiedoilla.

Käyttöliittymän toteutetaan suomeksi.

5.2 Navigointi

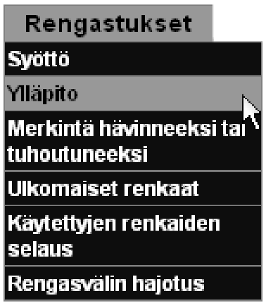
5.2.1 Valikko

Valikko (kuva 4) on monista sovelluksista tuttu alasvetovalikko, jonka avulla käyttäjä pystyy siirtymään mistä tahansa näytöstä suoraan mihin tahansa muuhun näyttöön. Valikkopalkki sijaitsee koko ajan sivun yläreunassa. Kun käyttäjä liikuttaa

Kuva 4: Ruudun yläreunassa sijaitseva valikkopalkki.

hiiren osoitinta valikon päällä, niin senhetkinen valinta korostetaan muuttamalla väriä ja palkin alle ilmestyy alavalikko.

Alavalikkossa (kuva 5) korostetaan valittu toiminto samalla tavalla, ja käyttäjä valitsee toiminnon klikkaamalla sitä. Jos käyttäjä ei haluakaan valita mitään, niin hän siirtää hiiren valikon ulkopuolelle, jolloin alavalikko häviää.

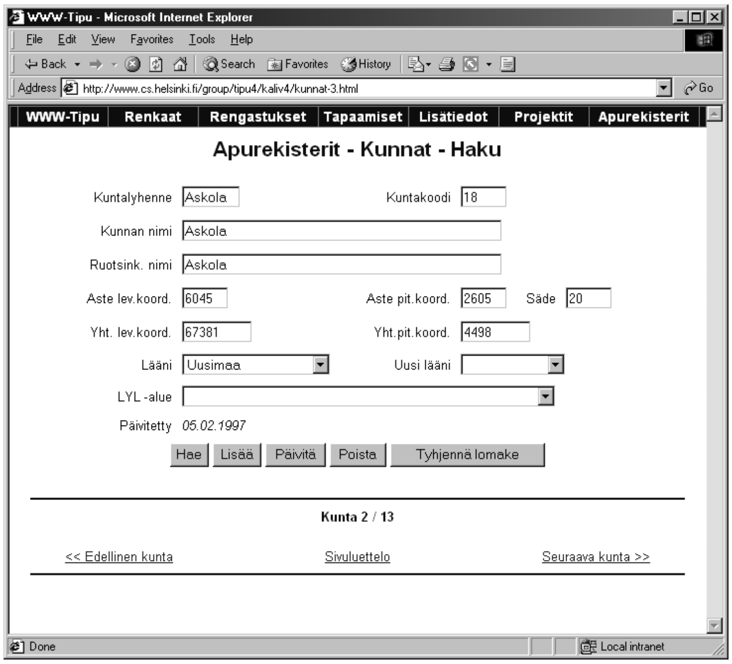


Kuva 5: Alavalikko rengastustoiminnoista.

5.2.2 Lomake

Suurin osa WWW-Tipun näytöistä ovat lomakkeita joilla katsotaan tai muokataan tietokannan sisältöä (esim. kuva 6 sivulla 33). Lomakkeelle syötettyjen tietojen tarkistus tapahtuu vasta kun se 'lähetetään', eli yritetään lisätä tai päivittää tietoja tietokantaan. Ainoastaan yksinkertaiset tarkistukset, kuten syötteen pituus, tehdään tietoja syötettäessä.

Lomake koostuu kentistä, joiden arvo syötetään kentän mahdollisesta arvojoukosta riippuen eri tavoin, joko tekstikenttillä ja valintalaatikoilla. Valintalaatikkoa käytetään, jos kentän mahdollisia arvoja on vähän. Niiden tekstikenttien vieressä joiden arvojoukko on rajattu mutta suuri, on nappi jota painamalla pääsee selailemaan mahdollisia arvoja uudessa ikkunassa ja valitsemaan niistä yhden.



Kuva 6: Tyypillinen näkymä käyttäjän selatessa kantaan talletettujen kuntien tietoja.

5.2.3 Painikkeet

Painikkeet sijaitsevat lyhyessä lomakkeessa sen lopussa ja pitkässä sekä alussa että lopussa.

Painike	Toiminto
Hae	Suorittaa haun lomakkeella olevilla tiedoilla. Tulokset näytetään listassa linkkeinä joita klikkaamalla kyseisen tietueen arvot tuodaan lomakkeelle.
Lisää	Lisätään lomakkeella olevat tiedot kantaan. Jos niissä on virheitä ilmoitetaan siitä käyttäjälle.
Päivitä	Päivitetään lomakkeella olevat tiedot kantaan. Jos niissä on virheitä ilmoitetaan siitä käyttäjälle.

Poista	Poistetaan lomakkeilla olevilla tiedoilla löytyvä tietue tietokannasta. Jos ei löydy tasan yhtä tietuetta ilmoitetaan siitä käyttäjälle eikä poisteta mitään. Jokainen poisto varmistetaan käyttäjältä kysymällä häneltä onko hän varma suoritettavasta poistosta.
Tyhjennä lomake	Tyhjennetään kaikki lomakkeella olevat kentät.

5.2.4 Linkit

Jos on tehty haku ja tuloksia on enemmän kuin yksi niin niitä pystytään selailemaan linkeillä “« Edellinen” ja “Seuraava »”. Linkkiä “Sivuluettelo” seuraamaalla päästään näkemään hakutulokset luettelomuodossa. Linkit sijaitsevat painikkeiden alapuolella.

5.2.5 Virheilmoitukset

Virhetilanteen sattuessa käyttäjälle tulostetaan virheilmoitus sivulle, jolla käyttäjä oli virheen sattuessa. Virheilmoitus kertoo virheen syyn. Virhetilanteen jälkeen käyttäjä voi korjata virheellisen lomakkeen tai tietokantavirheen sattuessa toistaa kesken jääneen operaation.

5.3 Esimerkki käyttötapauksesta

Rengastustoimiston työntekijä N.N. haluaa tarkistaa renkaalla X rengastetun linnun tiedot.

1. Selaimen käynnistäminen. N.N. käynnistää selainohjelmansa ja avaa selaimeen WWW-Tipun aloitussivun.

2. Sisään kirjautuminen. Hän syöttää tunnuksen ja salasanan ja klikkaa "Kirjaudu sisään"-painiketta.

3. Rengastusnäytölle siirtyminen. Hän valitsee alasvetovalikosta "Rengastukset" ja klikkaa sitä. Tämä vie hänet rengastukset näytölle.

4. Hakuehtojen syöttäminen. Hän syöttää renkaan tunnuksen X sille tarkoitettuun kenttään ja klikkaa "Hae"-painiketta.

5. Tietueen valitseminen. Annetuilla tiedoilla löytyi yksi rengastus joten N.N. pääsee suoraan tarkistamaan haluamansa tiedot ruudulle tulleesta lomakkeesta.

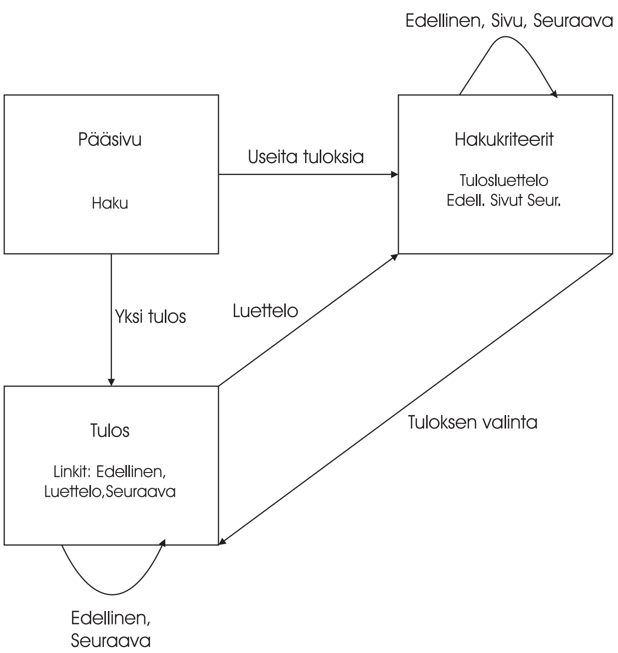
5.4 Tiedon haku

Käyttöliittymässä on tarpeen käyttää hakutoimintoja kahteen eri tarkoitukseen: kokonaisten tietueiden tietojen ja yksittäisten kenttien arvojen hakemiseen tietokannasta.

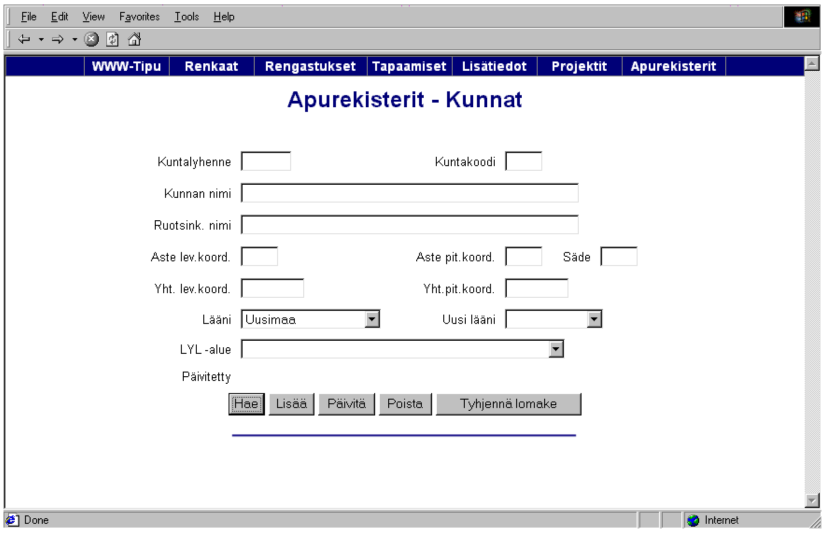
5.4.1 Kokonaisten tietueiden haku

Kokonaisilla tietueilla tarkoitetaan yhdellä sivulla kerrallaan näkyvää tietokokonaisuutta, esimerkiksi Apurekisterit - Kunnat -sivun kenttien sisältöä. Hakutoimintoa tarvitaan tällöin yhden tai useamman hakuehdot täyttävän tietueen tietojen tarkastelua, ylläpitoa tai kopiointia varten. Haun toimintaperiaate ilmenee kuvasta 7.

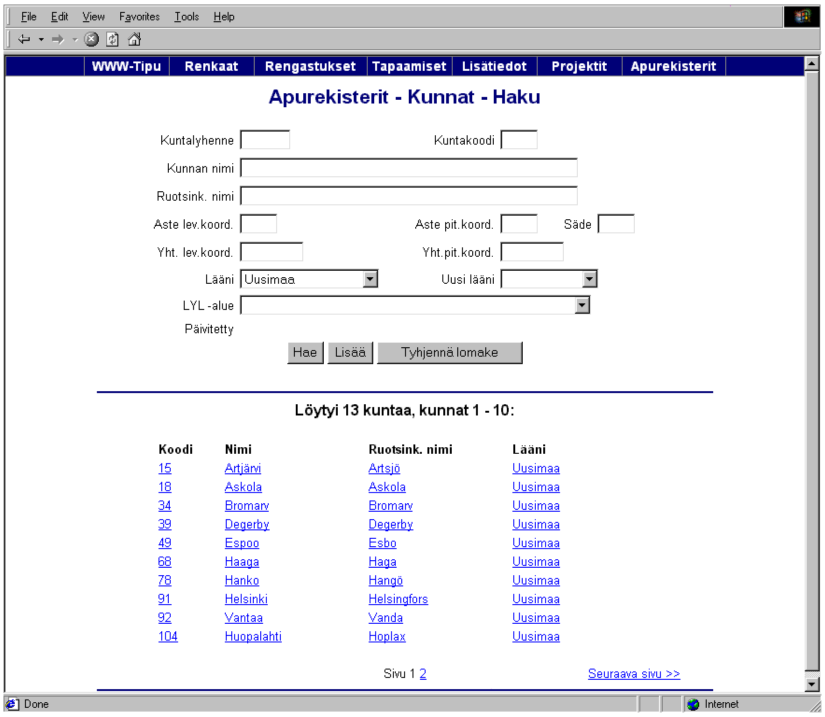
Hakutoiminto on käytettävissä kaikissa syöttö- ja ylläpitolomakkeissa. Kun käyttäjä on kirjoittanut haluamansa hakukriteerit lomakkeeseen, hän voi aloittaa haun napsauttamalla Hae-painiketta (kuva 8).



Kuva 7: Hakutapahtuman kulku. Pääsivusta on esimerkkinä kuva 8, hakukriteereistä kuva 9 ja tulossivusta kuva 10.



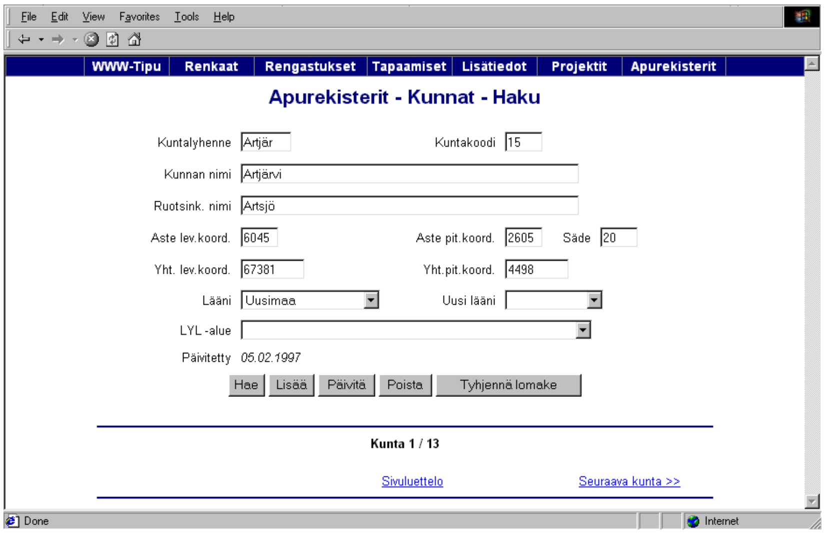
Kuva 8: Kuntien ylläpitolomake. Käyttäjä pääsee Hae-painikkeella hakemaan kuntia.



Kuva 9: Kuntien hakulomake. Lomakkeella on suoritettu haku kaikista Uudenmaan kunnista ja tulokset ovat listattuna hakukenttien alla.

Jos haku ei palauta yhtään tulosta, tulossivulla näkyy siitä ilmoitus. Jos haku palauttaa vain yhden tuloksen, sitä vastaava tietue näytetään käyttäjälle automaattisesti omalla sivullaan. Jos hakutuloksia on vähintään kaksi, hakusivun alalaidassa näytetään kunkin kriteerejä vastaavan tietueen avaintiedot kymmenen kappaleen luetteloissa (kuva 9). Käyttäjä voi selata luetteloa eteen- ja taaksepäin tai siirtyä suoraan haluamalleen sivulle tulosluettelossa.

Käyttäjä voi valita yksittäisen tietueen muokattavaksi valitsemalla sen luettelosta. Tällöin valittu tietue näytetään samoin kuin tavallisessa syöttölomakkeessa (kuva 10). Sivun alaosassa on linkki tulosluetteloon sekä edelliseen ja seuraavaan tietueeseen, mikäli ne ovat tulosjoukossa.



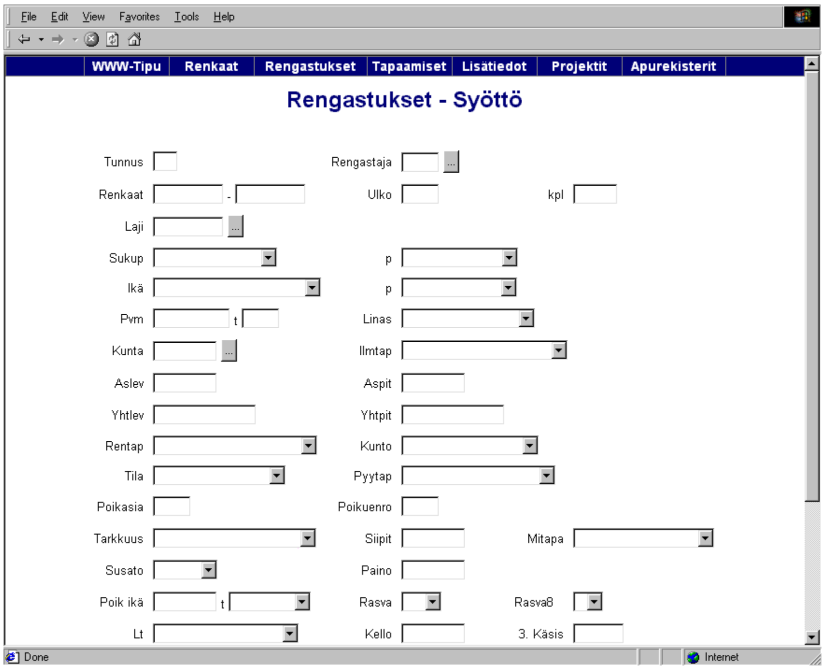
Kuva 10: Hakutulosten selaus tietueittain.

5.4.2 Yksittäisten kenttien arvojen haku

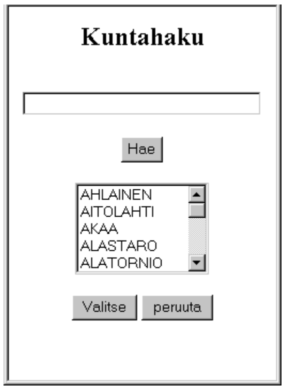
Monissa lomakkeissa on kenttiä, joiden kelvolliset arvot voidaan lukea tietokannasta. Näitä arvoja vastaavat kuvaukset on yleensä järkevää tarjota käyttäjälle valintaluetteloissa. Muutamissa tapauksissa valintaluettelosta tulisi liian pitkä, jotta se olisi käyttökelpoinen WWW-lomakkeessa. Tämän vuoksi käyttöliittymä antaa käyttäjälle mahdollisuuden kirjoittaa kentän arvon itse tai hakea sen kenttään erillisellä hakutoiminnolla.

Kenttäkohtaisen arvon hakutoiminnon voi käynnistää hakupainikkeella, jonka tunnisteena on kolme pistettä (. . .). Painike on käytettävissä esimerkiksi Rengastukset - Syöttö -sivun Kunta-kentässä (kuva 11). Painiketta napsautettaessa kuvaruutuun avautuu uusi, pienempi ikkuna (kuva 12).

Tämä ikkuna sisältää kuvauksen haettavasta tiedosta, syöttökentän hakukriteerille, luettelon hakukriteeriä vastaavista tietokannan arvoista sekä painikkeet haun aloitukseen, haun peruutukseen ja arvon valintaan pääikkunan syöttökenttään. Ikkuna sulkeutuu automaattisesti, kun käyttäjä valitsee Valitse- tai Peruuta-painikkeen.



Kuva 11: Rengastustietolomakkeessa joillakin kentillä on oma hakupainikkeensa.



Kuva 12: Mikäli esim. Kunnan lyhennettä ei muista, voi arvoa hakea kenttäkohtaisella hakutoiminnolla.

6 Rajoitteet suunnittelulle ja toteutukselle

6.1 Noudatettavat standardit

Servlet-ohjelmamoduuli kommunikoi käyttäjän kanssa HTTP 1.1 -protokollan [4] mukaisesti. Asiakaspäässä näkyvät HTML-dokumentit kirjoitetaan HTML 4.01 -spesifikaation [3] mukaisesti.

6.2 Muut tarvittavat ohjelmat

WWW-Tipu tarvitsee toimiakseen Oracle 8 -tietokannan sekä rengastustietoja tarkistavan Check-ohjelman.

6.3 Arvio toteutuksen koosta

Varovainen esiarvio toteutuksen koosta on noin 2100 riviä, noin 30 rivin tarkkuudella. (Laskettu Ohjelmistotuotanto-kurssin kohdan "arviointi"[7] avulla.)

6.4 Ohjelmointityyli

WWW-Tipu kirjoitetaan Sun Microsystemsin kehittämällä Java 1.2 -kielellä. Kielen syntaktinen ulkoasu (mm. sisennystyyli, isojen ja pienten kirjainten käyttö) on Java Code Conventions -spesifikaation [1] mukainen.

Luokkien, metodien ja muuttujien nimet kirjoitetaan englanniksi, jotta niistä tulisi johdonmukaisia Java-kielen varattujen sanojen ja valmiiden luokkakirjastojen nimien kanssa. Tulostettava teksti ja kommentit kirjoitetaan suomeksi, jotta ne olisivat yhdenmukaisia aikaisemman järjestelmän tulostamien tekstien kanssa.

7 Testaussuunnitelma

Tässä kuvataan yleisesti testauksen eri vaiheet. Tarkemmat kuvaukset ja testitapaukset tulevat erilliseen testausdokumenttiin.

7.1 Testimoottori

Testattavat moduulit (poislukien servletit) sisältävät main-metodin, jolla sisäistä toiminnallisuutta, esimerkiksi tietokantayhteyksiä voidaan testata tarvittavilla suorituspoluilla ja muuttujilla.

Servletejä voidaan testata ja kuormittaa erillisellä luokalla, joka lähettää servletille HTTP GET-muotoisia pyyntöjä ja tallettaa servletin vastaukset tekstitiedostoihin.

7.2 Komponenttitestaus

Ohjelmiston jokainen moduuli (ks. kappale 4) testataan erillisenä komponenttina. Komponentit testataan “white box”- menetelmällä, jotta voidaan testata niiden sisäistä toimivuutta. Testauksessa on tarkoitus käyttää polkukattavuutta, eli jokainen mahdollinen suorituspolku testataan ainakin kerran. Jokaiselle komponentille määrätään erityinen testaaja, joka on eri henkilö kuin kyseisen komponentin implementoija.

7.3 Integrointitestaus

Integrointitestauksessa varmistetaan järjestelmän eri moduulien (luokkien) toiminta yhdessä mustalaatikkotestauksena. Kunkin luokan syötteiden ja tulosteiden oikeellisuus varmistetaan jakamalla niiden arvoalueet ekvivalenssiluokkiin, joista muodostetaan testiaineisto. Tämän jälkeen toteutetaan luokille testiaineiston mukaiset testitapaukset.

WWW-Tipu testataan alhaalta ylös-menetelmällä. Tällöin ohjelmiston toteutus ja testaus aloitetaan luokkakaavion alimmista luokista. Järjestelmästä vielä puuttuvien alempia luokkia kutsuvien luokkien toimintaa simuloidaan erikseen koodattavilla ajuriluokilla. Näiden ajuriluokkien tulosteista voidaan todentaa kutsutun luokan tulosteen vastaavuus määriteltyyn.

7.4 Validointitestaus

Validointitestaus tehdään kun integrointitestaus on valmis ja siinä havaitut virheet korjattu. Validointitestauksen tarkoituksena on tarkastaa täyttääkö WWW-Tipu sille vaatimusmäärittelyssä asetetut vaatimukset. Tämä tapahtuu suorittamalla joukko testitapauksia “black box”-menetelmällä, jotka osoittavat että ohjelmisto täyttää sille asetetut toiminnallisuus-, suorituskyky- ja muut vaatimukset sekä sen että dokumentaatio on oikeellinen.

7.5 Hyväksymistestaus

Viimeisenä suoritetaan hyväksymistestaus, tämä on tilaisuus jossa loppukäyttäjät käyttävät WWW-Tipua oikealla aineistolla lopulisessa “tuotantoympäristössä”. Tässä käydään asiakkaan kanssa läpi ohjelmisto, dokumentointi ja käyttöohje ja katsotaan vastaako se hänen odotuksiaan.

Viitteet

[1] Code conventions for the java programming language. http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html.

[2] Freemarker javadoc (package `freemarker.tempalte`). http://freemarker.sourceforge.net/docs/.

[3] Hypertext markup language 4.01 specification. http://www.w3.org/TR/html401/.

[4] Hypertext transfer protocol 1.1 specification. http://www.ietf.org/rfc/rfc2616.txt.

[5] Java 2 platform, standard edition, v 1.3 api specification. http://java.sun.com/products/jdk/1.3/docs/api/.

[6] Java database connectivity (jdbc) 2.0 specification. http://java.sun.com/products/jdbc/.

[7] Ohjelmistotuotantoprojekti, syksy 2000, arviointi. http://www.cs.helsinki.fi/u/taina/ohtu/s-2000/luennot/arviointi/.

[8] Benjamin Geer. Freemarker manual. http://freemarker.sourceforge.net/docs/manual/.

[9] Danny Coward James Duncan Davidson. Java servlet specification 2.2. http://java.sun.com/products/servlet/.